Лабораторная работа 1

Характеристики языка:

Вариант 23 ЯПИС:

1.Требования к разрабатываемому языку

1. **Встроенные типы не менее трех:**
   table, row, column, numb, string.
2. **Возможность инициализация переменных всех типов при объявлении: <тип> <имя_переменной> = <выражение>**
   1. |Инициализирующее выражение может быть константным:
      const table t;
3. **Встроенные операции не менее 10 штук:**
   |<object1> = <object2> - операция присваивания
   <object1> ==/!= <object2> - операция сравнения объектов поэлементно
   | <numb/row/column/string> </>/<=/>= <numb/row/column/string> - операция сравнения значений либо длин строк либо длин столбцов или row.
   |<table1> + <table2> - объединение таблиц по вертикали, при равном кол. столбцов (+=)
   <table1> * <table2> - объединение таблиц по горизонтали, при равном кол. Строк (*=)
   <row1> + <row2> - объединение строк
   <column1> + <column2> - объединение колонок
   <table> + <column> - добавление колонки в таблицу, при условии, что размеры совпадают
   <table> + <row> - добавление строки в таблицу, при условии, что размеры совпадают
   <table> / <numb> - в таблице остаётся первые length(table) div numb строк
   <table> // <numb> - в таблице остаётся первые width(table) div numb столбцов
   <row> / <numb> - оставляет первые length(row) div numb элементов
   <column> / <numb> - оставляет первые with(column) div numb элементов
   <column>[numb1][numb2] – находит элемент с numb1 numb2 индексами в таблице
   <row>[numb1] – находит элемент с numb1 индексом в строке
   <column>[numb2] – находит элемент в numb2 индексом в строке
   ++<numb1> - постфиксный инкремент
   --<numb1> - постфиксный декремент
4. **Встроенные функции**

1. Встроенные функции ввода\вывода для работы со встроенными типами read() write():
2. read_string() - поток для чтения строки
3. read_numb() - поток для чтения числа
4. print(<object>) - поток вывода объекта

5. **|Использование сложных выражений (составных и со скобками)**
6. **|Блочный оператор {}**
7. **Управляющие структуры**
    1. |Условный оператор (if-else): If() {} else if {} else {};
    2. Операторы цикла (while и until): while(cond) {}; {} until();
    3. |Оператор цикла с итерациями (for): for(i1, i2, ++i1);
8. **Пользовательские подпрограммы**
    1. |Передача и возврат параметров
    2. |Задание локальной и глобальной области видимости для имен переменных

3. Язык для работы с реляционными данными:

|*Встроенные типы: table, row, column, numb, string.*

|*Встроенные функции задания структур:*

create_table(numb1, numb2, [string/numb…]) - создать таблицу с данными размерами и данными данными.

create_row(numb1, [string/numb..]) - создать строку с размером num1 и данными данными.

create_column(numb1, [string/numb…]) - создать колонку с размером num2 и данными данными.

изменения данных:

изменение данных может происходить через операторы перечислисленные выше.

reshape(<table>, <numb1>, <num2>) - изменяет размеры таблицы.

del(<row/column>, <numb>) - удаляет элемент и возвращает объект

del_*col(<table>, <numb>)* - удалят колонку и возвращает таблицу

del_*row(<table>)* - удаляет строку и возвращает таблицу

insert(<column/row>, <string/numb>, numb) – вставляет элемент на numb индекс

Поиска:

find(<table/row/column>, <string/numb>) - возвращает индекс(-ы) для искомого значения либо NAN.

max(<table/row/column>) - возвращает макс. значение элемента типа numb

min(<table/row/column>) - возвращает мин. значение элемента типа numb

maxlen(<table/row/column>) - возвращает строку с максимальной длинной

minlen(<table/row/column>) - возвращает строку с минимальной длинной

4. **Свойства языка:**

|Объявление переменных : Явное

Преобразование типов : Не Явное

|Оператор присваивания : Многоцелевой, например, a, b = c, d

|Структуры, ограничивающие область видимости : Подпрограммы

|Маркер блочного оператора : Явные, например, { } или begin end

|Условные операторы : Двух вариантный оператор и многовариантный switch-case

|Перегрузка подпрограмм : Присутствует

|Передача параметров в подпрограмму : Только по значению и возвращаемому значению

|Допустимое место объявления подпрограмм: В начале программы

5. **Целевой код:**

Исполняемый файл (.exe), для генерации целевого кода использовать LLVM (http://llvm.org)

Фрагменты:

Написать 3 примера фрагментов кода языка по варианту, все свойства должны быть использованы, встроенные типы, операции и функции должны быть использованы и показаны на примерах.

1. Найти среднее значение численной строки.

```
##################################################################
numb function mean(row row1)
{
    numb sum = 0;
    for(numb I=0; numb < length(row1); ++I){
        sum += row1[I]
```

```
        }
        return sum / length(row1);
}
row row1 = create_row(3, [1, 4.5, 6]);
print(mean(row1));
```

##

2. Фрагмент кода,

##

```
table function add(table t, row r){
        if len(r) != width(t){
                print(“Incorrect shapes”);
                return t
        }
        else{
        return t + r;
        }
}
table function add(table t,  column c){
        if len(c) != len(t){
                print(“Incorrect shapes”);
                return t;
        }
        else{
        return t + c;
        }
}
table t = create_table(2, 2);
column c = create_column(2);
```

```
row r = create_row(3);
numb I = 0;
numb j = 0;
while(i < 2){
        while(j < 2){
                string val = read_string();
                t[i][j] = val;
                ++j;
        }
        ++i;
}
I = 0;
{
        r[i] = read_string();
        ++i;
} until (I < 3);
c[0] = read_string();
c[1] = read_string();
table res1, res2;
res1, res2 = add(t, r), add(t, c);
print(res1);
print(res2);
```

##########

3. Фрагмент кода, в котором тестируются все функции, из 1.1. и 1.2, которые ещё не были использованы выше на тестовом столбце строке и таблице.

##########

```
table t = create_table(3, 3, [1, 2, “hello”, 4, 0];
# [[1, 2, “hello”], [4, 0, None], [None, None, None]]
row r = create_row(3, [“T”, 0, -5.6]);
```

```
column c = create_column(3, [“C”, “A”, “D”];
row r1 = create_row(3, [“T”, 0, -5.6]);
if (r == r1) {print(“True”);}  # ~~~ True
r1[1] = “Hell”;
if (r != r1) { print(“True”);} # ~~~ True
table t1 = copy(t);   # копируем t в t1
print(t + t1);   # ~~~ будет [[1, 2, “hello”], [4, 0, None], [None, None, None], [1, 2,
                 #                “hello”], [4, 0, None], [None, None, None]]
print(t * t1);   # ~~~ будет [[1, 2, “hello”, 1, 2, “hello”], [ 4, 0, None,  4, 0, None],
                          #          [ None, None, None,  None, None, None]]
print(r1 + r);  # [“T”, 0, -5.6, “T”, “Hell”, -5.6]
print(c + c);   # [“C”, “A”, “D”, “C”, “A”, “D”]
print(c + c == c * 2);  # ~~ True
print(t + c);  # ~~  [[1, 2, “hello”, “C”], [4, 0, None, “A”], [None, None, None, “D”]]
print(t + r);  # [[1, 2, “hello”], [4, 0, None], [None, None, None], [“T”, 0, -5.6]]
print(t / 2);  #  [[1, 2, “hello”]]
print(t // 2);   # [[1], [4], [None]]
print(r / 2);   # [“T”]
print(r[0][0] == r / 2);  # ~~ True
print(t[0]); # ~~ [[1, 2, “hello”]]
print(t[][0]);    # [[1], [4], [None]]
#### встроенные функции
print(reshape(t, 6, 1);  #  [[1, 2, “hello”, 4, 0, None, None, None, None]]
print(del_col(t, 0));    # [[1, 2], [4, 0], [None, None]]
print(del_row(t, 0));   # [[4, 0, None], [None, None, None]]
print(del(r, 0));          #  [0, -5.6]
print(insert(r, 1, 4));    #  [“T”, 0, -5.6, 1]
print(max(t), min(t), maxlen(t), minlen(t));    # 4 0 “hello” “hello”
print(find(c, “A”));   #   1
```

##